APPENDIX A

Inventor: Everett W. Stoub

Title: System for Converting Scrolling Display To Non-Scrolling Columnar Display

```
1   /*
2    *  %W% %E% Everett Stoub
3    *
4    *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
5    *
6    *  ez_text applet displays snaking text with in-line graphics
7    *      and produces a reader screen frame when clicked
8    */
9
10  /**
11   *  @author Everett Stoub %I%, %G%
12   */
13
14  import java.awt.*;
15  import java.applet.*;
16  import java.net.*;
17  import java.io.*;
18  import java.util.*;
19
20  public class ez_text extends Applet
21  {
22      private final boolean formattedText     = true;     //  if true, in-line graphics
23  and pull quotes are enabled
24
25      private String          title           = null;
26      private String          article         = null;
27      private ezrpnlap        panel           = null;
28      private ezrfrmap        frame           = null;
29
30      private MediaTracker    tracker         = null;
31      private Vector          imageFileNames  = null;     //  vector of all image file
32  names for document
33      private Vector          graphicsURLs    = null;     //  vector of all graphic
34  image URLs for document
35      private Vector          graphicsImages  = null;     //  vector of all graphic
36  images for document
37
38      private final boolean   frameOption     = true;     //  set false for 1Kb code
39  reduction
40
41      private final int       NONE            = 0;
42      private final int       LINE            = 1;
43      private final int       FLAT            = 2;
44      private final int       RAISED          = 3;
45      private final int       DROPPED         = 4;
46      private final int       ETCHED          = 5;
47
48      private int             frameType       = NONE;
49      private int             frameSize       = 0;
50
51      private final String    AG = "`";
52      private final String    AM = "&";
53      private final String    SC = ";";
54      private final String    SP = " ";
55      private final String    PL = "<p";
56      private final String    QL = "\"";
57      private final String    PR = "<p>";
58      private final String    QT = "quot";
59
60      private final String    alert_0 = "PLEASE READ - A Java ";
61      private final String    alert_1 = "Exception has occurred. ";
62      private final String    alert_2 = "Most likely, there has been a transmission
63  failure with one or more applet files. A Refresh or Reload command on your browser
64  will sometimes lead to a successful download. ";
65      private final String    alert_3 = "An attempt to read a local file has been
66  detected! In order to read this article, you must download the applet using hyper-text
67  transport protocols (http://...) from a web server. Please try again using http with a
68  valid URL. ";
69      private final String    alert_4 = "If the problem persists, please contact the Web
70  Master.";
```

```
1
2       public final void dbg(String s)
3       {
4           if (debug) System.out.println(s);
5       }
6       private boolean debug = false;      //  Print debugging info?
7
8       public String getTitle()
9       {
10      //  dbg("ezrpnlap.getTitle()");
11
12          return this.title;
13      }
14
15      public final void paint(Graphics g)
16      {
17      //  dbg("ez_text.paint(): frameType ="+frameType);
18
19          if (frameOption)
20          {
21              int i;
22              Dimension d = size();
23              switch (frameType)
24              {
25                  case LINE:
26                  {   g.setColor(getForeground());
27                      for (i = 0; i < frameSize; i++)
28                          g.drawLine(0,d.height-i-1,d.width-i-1,d.height-i-1);
29                  }   break;
30                  case FLAT:
31                  {   g.setColor(getForeground());
32                      for (i = 0; i < frameSize; i++)
33                          g.drawRect(i,i,d.width-2*i-1,d.height-2*i-1);
34                  }   break;
35                  case RAISED:
36                  case DROPPED:
37                  case ETCHED:
38                  {   boolean raise = frameType==RAISED;
39                      int R = (getForeground().getRed()+  getBackground().getRed()  )/2;
40                      int G = (getForeground().getGreen()+getBackground().getGreen())/2;
41                      int B = (getForeground().getBlue()+ getBackground().getBlue() )/2;
42                      g.setColor(new Color(R,G,B));
43                      for (i = 0; i < frameSize; i++)
44                      {
45                          if (frameType == ETCHED) raise = i >= frameSize/2;
46                          g.draw3DRect(i,i,d.width-2*i-1,d.height-2*i-1,raise);
47                      }
48                      if (frameType == ETCHED)
49                      {
50                          g.setColor(getForeground());
51                          g.drawLine(0,0,frameSize/2,frameSize/2);
52                          g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
53  frameSize/2,d.height-frameSize/2);
54                          g.setColor(getBackground());
55                          g.drawLine(frameSize/2,frameSize/2,frameSize,frameSize);
56                          g.drawLine(d.width-frameSize/2,d.height-frameSize/2,d.width-
57  1,d.height-1);
58                      }
59                      else
60                      {
61
62      g.setColor((frameType==RAISED)?getBackground():getForeground());
63                          g.drawLine(0,0,frameSize,frameSize);
64
65      g.setColor((frameType==RAISED)?getForeground():getBackground());
66                          g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
67  1,d.height-1);
68                      }
69                  }
70              }
```

```
        }
    }

    public synchronized Image fetchImage(String fileName)
    {
    //  dbg("ez_text.fetchImage("+fileName+")");

        if (!formattedText || fileName == null)
        {
            return null;
        }
        else
        {
            Image theImage = null;

            int imageDatabaseIndex =                        //  get reference to image
database file name
                imageFileNames.indexOf(fileName);

            if (imageDatabaseIndex < 0)                     //  not registered yet:
add it in
            {
                URL theURL = null;

                try                                         //  look in the
documentBase
                {
                    theURL = new URL(getDocumentBase(), fileName);
                    InputStream is = theURL.openStream();   //  test connection
                    theImage = getImage(theURL);
                    tracker.addImage(theImage,0);

                //  jump-start image loading

                    tracker.checkID(0,true);
                    try   { tracker.waitForID(0);       }
                    catch ( InterruptedException ie ) { }
                    Image bufferedImage = createImage(1,1);
                    if (bufferedImage != null)
                    {
                        Graphics bg = bufferedImage.getGraphics();
                        if (bg != null)
                        {
                            bg.drawImage(theImage,0,0,this);
                            bg.dispose();
                        }
                    }
                    bufferedImage.flush();
                }
                catch (MalformedURLException u)
                {
                    System.out.println("ez_text.fetchImage("+fileName+"):
MalformedURLException = "+u);
                    theImage = null;
                }
                catch (IOException io)
                {
                    System.out.println("ez_text.fetchImage("+fileName+"): InputStream
IOException = "+io);
                    theImage = null;
                }

            //  update image database for document

                imageFileNames.addElement(fileName);        //  the filename is the
main key to image database access
                graphicsURLs.addElement(theURL);            //  in case we need to
download it again later
                graphicsImages.addElement(theImage);        //  this may be reset to
null with imageUpdate
```

```
            }
            else
            {
                theImage = (Image)graphicsImages.elementAt(imageDatabaseIndex);
            }

            return theImage;
        }
    }

    public void disposeFrame()
    {
    //  dbg("ez_text.disposeFrame()");

        if (frame != null)
        {
            Event evt = new Event(frame, Event.WINDOW_DESTROY, null);
            Component co[] = frame.getComponents();
            for (int i = 0; i < co.length; i++)
                co[i].handleEvent(evt);

            frame.dispose();
            frame = null;
        }
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("ez_text.mouseUp(Event "+evt+", int "+x+", int "+y+") ");

        if (frame != null)
            frame.toFront();
        else
            frame = new ezrfrmap(title, article, this);

        return false;
    }

    public String htmlCanon(String itsText, boolean convertNWSs)
    {
    //  dbg("ezrpnlap.htmlCanon(\n--------------"+itsText+"\n--------------,
"+convertNWSs+")");

        if (!formattedText)
            return itsText;

        if (convertNWSs)                                    //  look for
[`title<p]...
        {
            String LCtext = itsText.toLowerCase();

            if (LCtext.startsWith(AG))                      //  initial accent
grave specifies article title
            {
                int endTitle = LCtext.indexOf(PL);          //  end of line after
canonization
                if (endTitle > 0)
                {
                    title = itsText.substring(1,endTitle);
                    itsText = itsText.substring(endTitle);
                }
            }
        }
        else
        {
        //  replace character entities with spaces (except quot -> \")

            StringTokenizer entities = new StringTokenizer(itsText,AM,true);
            StringBuffer newText = new StringBuffer(70);

```

```
            while (entities.hasMoreTokens())
            {
                String entity =                                 //  [body
text],[&][#nnn;body text],...,[&][#nnn;body text]
                    entities.nextToken(AM);

                if (entity.equals(AM))
                {
                    String eText =                              //  should be
character entity text #nnn or name
                        entities.nextToken(SC);

                    entity = entities.nextToken();              //  should be ";" if
eText is valid character entity value

                    if (entity.equals(SC))                      //  replace character
entity with local character (set)
                    {
                        if (eText.equals(QT))
                        {
                            newText.append(QL);
                        }
                        else
                        {
                            newText.append(SP);
                        }
                    }
                    else                                        //  syntax error:
                    {
                        newText.append(AM).append(eText).append(entity);
                    }
                }
                else
                {
                    newText.append(entity);                     //  standard text
                }
            }
            itsText = newText.toString();

        //  look for [<title...>title text</title...]

            String LCtext = itsText.toLowerCase();

            int ending = 0, starts = LCtext.indexOf("<title");
            if (starts > -1)
            {
                starts = LCtext.indexOf(">",starts)+1;          //  start of title
                if (starts > 0 && starts < LCtext.length())
                {
                    ending = LCtext.indexOf("</title",starts);
                    if (ending > -1)
                    {
                        title = itsText.substring(starts,ending);
                    }
                }
            }

        //  look for [<body...>body text</body...]

            starts = LCtext.indexOf("<body");
            if (starts > -1)
            {
                starts = LCtext.indexOf(">",starts)+1;          //  start of body
                if (starts > 0 && starts < LCtext.length())
                {
                    ending = LCtext.indexOf("</body",starts);
                    if (ending > -1)
                    {
                        return itsText.substring(starts,ending);
                    }
```

```
                }
            }
        }
        return itsText;
    }

    public String lineCanon(String itsText, boolean convertCRs)
    {
    //  dbg("ez_text.lineCanon()");

        if (itsText == null
        ||  itsText.length() == 0)  return itsText;

        String newSeparator = convertCRs?PR:SP;                 //  target return code
        String searchSeparators[] =                             //  replacable return
codes
            {
                "\n"+"\r",                                      //  chr(10), chr(13)
                "\r"+"\n",                                      //  chr(13), chr(10)

                "\r",                                           //  chr(13)
                "\n",                                           //  chr(10)
                "\f"                                            //  chr(12)
            };
        String newText = new String(itsText);

        for (int i = 0; i < searchSeparators.length; i++)       //  replace old with
new
        {
            String oldSeparator = new String(searchSeparators[i]);
            newText = replaceString(newText,oldSeparator,newSeparator);
        }
        return newText;
    }

    public String replaceString
    (   String theString                                        //  original string
    ,   String oldString                                        //  replace-ee
    ,   String newString                                        //  replace-or
    )
    {
    //  dbg("ez_Text.replaceString(theString, old: "+oldString+", new:
"+newString+")");

        StringBuffer out = new StringBuffer();

        int oldIndex = -1, oldEnd = 0;
        int oldLength = oldString.length();

        while ((oldIndex = theString.indexOf(oldString, oldEnd)) > -1)
        {
            String piece = theString.substring(oldEnd,oldIndex);//  got it? get it!
good...
            out.append(piece);                                  //  add the piece
before the next old string
            out.append(newString);                              //  get the
replacement
            oldEnd = oldIndex + oldLength;                      //  skip to the end of
this old string
        }                                                       //  nothing more to
replace
        if (oldEnd < theString.length())
        {
            String piece = theString.substring(oldEnd);
            out.append(piece);                                  //  add the piece
after the last old string
        }
        return out.toString();
    }

```

```
1       public Insets insets()
2       {
3       //  dbg("ezrpnlap.insets()");
4
5           if (frameOption) switch (frameType)
6           {
7               case LINE:
8                   return new Insets(0,0,frameSize,0);
9               case FLAT:
10              case RAISED:
11              case DROPPED:
12              case ETCHED:
13                  return new Insets(frameSize,frameSize,frameSize,frameSize);
14          }
15          return new Insets(0,0,0,0);
16      }
17
18      public void init()
19      {
20      //  dbg("ez_text.init<>: parent background = "+getParent().getBackground());
21
22          if (formattedText)
23          {
24              imageFileNames  = new Vector(10);                   //  must be created
25  before calls to fetchImage
26              graphicsURLs    = new Vector(10);                   //  must be created
27  before calls to fetchImage
28              graphicsImages  = new Vector(10);                   //  must be created
29  before calls to fetchImage
30          }
31
32          tracker = new MediaTracker(this);
33
34          String s = getParameter("fileName");
35          if (s != null) try                                      //  new URL, etc.
36          {
37              URL url = new URL(getDocumentBase(), s);
38              InputStream is = url.openStream();
39              BufferedInputStream bis = new BufferedInputStream(is);
40
41              int len = 0;
42              byte buffer[] = new byte[512];
43              StringBuffer pageText = new StringBuffer();
44              while ((len = bis.read(buffer)) > 0)
45                  pageText.append(new String(buffer,0,0,len));
46
47              bis.close();
48              is.close();
49
50              boolean toHTML = s.toLowerCase().endsWith(".txt") ||
51  s.toLowerCase().endsWith(".text") || s.toLowerCase().endsWith(".nws");
52              article = htmlCanon                                 //  retains only
53  supported tags
54              (   lineCanon(pageText.toString()
55              ,   toHTML)                                         //  true converts
56  'CR's to '<p>'s
57              ,   toHTML                                          //  true converts
58  enews tags to html
59              );                                                  //  strips off head &
60  saves title
61
62          //  start loading in-line graphics right away
63
64              StringTokenizer images = new StringTokenizer(article,"@",true);
65              while (images.hasMoreTokens())
66              {
67                  String token = images.nextToken();
68                  if (token.equals("@"))
69                  {
```

```
                    String fileName = images.nextToken("<");    //  <p> following
fileName
                    if (fileName.toLowerCase().endsWith(".jpeg")
                    ||  fileName.toLowerCase().endsWith(".gif"))
                        fetchImage(fileName.trim());
                }
            }
        }
        catch (MalformedURLException mu)
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Malformed URL
").append(alert_1).append(alert_2).append(alert_4);

            article = msg.toString();

            String e = "Could not open "+s;
            showStatus(e);
            System.out.println(e+"; exception: "+mu);
        }
        catch (IOException io)
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Input/Output
").append(alert_1).append(alert_2).append(alert_4);

            article = msg.toString();

            String e = "Could not open "+s;
            showStatus(e);
            System.out.println(e+"; exception: "+io);
        }
        catch (SecurityException se)
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Security
").append(alert_1).append(alert_3).append(alert_4);

            article = msg.toString();

            String e = "Could not open "+s;
            showStatus(e);
            System.out.println(e+"; exception: "+se);
        }
        else
        {
            StringBuffer msg = new StringBuffer();
            msg.append(alert_0).append("Input/Output ").append(alert_1);
            msg.append("No article file specification was detected! ");
            msg.append(alert_2).append(alert_4);

            article = msg.toString();

            String e = "No fileName specified";
            showStatus(e);
            System.out.println(e);
        }
        this.setLayout(new BorderLayout());
        this.setBackground(getParent().getBackground());
        this.setForeground(getParent().getForeground());

        panel = new ezrpnlap(article, this, false);                    //  NOT the reader
screen

    //  add insets for applet boundaries...
        this.add("Center", panel);

        if (frameOption)
        {
            String b = getParameter("frameType");
```

```
            if (b != null)
            {
                b = b.toUpperCase();
                if (b.equals("LINE"))    {   frameSize = 1;   frameType = LINE;    }
                if (b.equals("FLAT"))    {   frameSize = 1;   frameType = FLAT;    }
                if (b.equals("RAISED")) {   frameSize = 2;   frameType = RAISED;  }
                if (b.equals("DROPPED")){   frameSize = 2;   frameType = DROPPED;}
                if (b.equals("ETCHED")){    frameSize = 4;   frameType = ETCHED;  }
            }

            String w = getParameter("frameSize");
            if (w != null)
            try   {frameSize = Integer.parseInt(w); }
            catch (NumberFormatException nf)     {  }
        }
    }
}

/*
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  ez_text applet reader screen frame
 */

/**
 *  @author Everett Stoub %I%, %G%
 */

import java.awt.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.util.*;

public final class ezrfrmap extends Frame
{
    private ezrpnlap    readerScreen;
    private Panel       control;
    private ez_text     ez_Text;
//  private Label       footer;
    private boolean     demo = true;

    private String buttonName[] =
    {   "Home"
    ,   "Previous"
    ,   "Next"
    ,   "Font +"
    ,   "Font -"
    };

    public ezrfrmap(String title, String readerContent, ez_text ez_Text)
    {
        super(title);

    //  dbg("ezrfrmap.<init>");

        this.ez_Text = ez_Text;
        this.readerScreen = new ezrpnlap(readerContent, ez_Text, true);
        this.add("Center", readerScreen);

        control = new Panel();
        control.setBackground(Color.gray);
    //  control.setLayout(new FlowLayout(FlowLayout.RIGHT));
        control.setLayout(new FlowLayout(FlowLayout.CENTER));

        StringBuffer version = new StringBuffer();
        version.append("ION EZ Text ");
        if (demo)
```

```
            version.append("Demo");
        else
            version.append("v1.03");    // [An-major].[An-minor][An-tweak][Aa-fix]

        control.add(new Label(version.toString()));
        for (int i = 0; i < buttonName.length; i++)
        {
            Button b = new Button(buttonName[i]);
            b.setBackground(Color.white);
            control.add(b);
        }
        control.add(new Label("(c)'97 ION Systems, Inc. 888.ION.JAVA"));

        this.add("South", control);

        this.pack();
        this.show();
    }

    public boolean action(Event e, Object o)
    {
        if (e.target instanceof Button)
        {
            if (o == (buttonName[0]))   readerScreen.doViewerHome();        else
            if (o == (buttonName[1]))   readerScreen.doViewerLeft();        else
            if (o == (buttonName[2]))   readerScreen.doViewerRight();       else
            if (o == (buttonName[3]))   readerScreen.doViewerSize(false);   else
            if (o == (buttonName[4]))   readerScreen.doViewerSize(true);    else
return false;
        //  repaint();
            return true;
        }
        else return false;
    }

/*  private final void dbg(String  s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = false;      //  Print debugging info?
*/
    public final boolean keyDown(Event evt, int key)    //  note:  keyUp is not
utilized at all (missing in JDK 1.02 on Mac)
    {
    //  dbg("ezrfrmap.keyDown("+evt+", "+key+")");

        boolean down = key == Event.DOWN;

        switch (key)
        {
        case Event.HOME:    readerScreen.doViewerHome();        repaint();  return
true;
        case Event.PGDN:
        case Event.RIGHT:   readerScreen.doViewerRight();       repaint();  return
true;
        case Event.PGUP:
        case Event.LEFT:    readerScreen.doViewerLeft();        repaint();  return
true;
        case Event.UP:
        case Event.DOWN:    readerScreen.doViewerSize(down);    repaint();  return
true;
        }

        return false;
    }

    public final boolean handleEvent(Event evt)
    {
    //  dbg("ezrfrmap.handleEvent("+evt+")");

```

```
        switch (evt.id)
        {
        case Event.WINDOW_EXPOSE:       readerScreen.requestFocus();    break;
        case Event.WINDOW_DEICONIFY:    this.show();                    break;
        case Event.WINDOW_ICONIFY:      this.hide();                    break;
        case Event.WINDOW_DESTROY:      ez_Text.disposeFrame();         break;
        }
        return super.handleEvent(evt);
    }

    public void update(Graphics g)
    {
    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
        g.setColor(getForeground());
        paint(g);
    }
}

/*
 *  %W% %E% Everett Stoub
 *
 *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
 *
 *  web crawler with ReaderScreen
 */

/**
 *  @author Everett Stoub %I%, %G%
 */

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.net.*;
import java.io.*;
import java.util.*;

public final class ezrpnlap extends Panel
{
    public final static int     LEFT        = 0;
    public final static int     CENTER      = 1;
    public final static int     RIGHT       = 2;
    public final static String  MT          = "";
    public final static String  BL          = " ";

    private boolean     demo                = true; //  mild display spam
    private final boolean formattedText     = true; //  if true, in-line graphics and
pull quotes are enabled

    private ez_text     ez_Text             = null;

    private String      readerContent       = null;
    private Image       readerScreen        = null;

    private int         readerFontSize[]    = { 8,
9,10,11,12,14,18,24,30,36,42,48,56,64,72,80,96};
    private int         readerFootIndex     = 2;    //  this 2-step decrement is also
hard coded in stepFontSize()
    private int         readerSizeIndex     = 4;    //  default for on-webPage article
rendering
    private int         readerPullIndex     = 5;    //  this 1-step increment is also
hard coded in stepFontSize()
    private int         readerHeadIndex     = 6;    //  this 2-step increment is also
hard coded in stepFontSize()
    private Font        readerFooterFont    = null;
    private Font        readerScreenFont    = null;
    private Font        readerPQuoteFont    = null;
    private Font        readerHeaderFont    = null;

```

```
    private int         readerAlignment     = LEFT;
    private int         readerPerimeter     = 10;
    private int         readerNumCols       = 1;
    private int         readerLeading       = 0;
    private int         readerPageNum       = 1;
    private int         readerNumPages      = 0;
    private int         readerEstPages      = 1;
    private int         readerEndIndex      = 0;
    private int         readerStartIndex    = 0;
    private int         readerTextLength    = 0;
    private boolean     readerScreenDirty   = true;     // true if panel needs redraw
    private boolean     readerFrame         = false;    // true if panel owned by
frame
    private boolean     readerHeader        = false;    // true if article begins with
`title

    private Color       readerBackColor     = new Color(231,231,231);
    private Vector      readerPageStart     = new Vector(50);
    private Rectangle   readerRect          = new Rectangle();

//  variable principally for paintTextArea

    private int         lowestTextAreaBaseLine = 0;
    private int         controlSize         = 0;
    private int         pullQuoteLineWidth  = 2;
    private int         pullQuoteSizeIncr   = 1;

    private boolean     drawPullUnderliner  = false;    //  switch to finish pull
quote display
    private boolean     finishPullQuoteText = false;    //  switch to finish pull
quote content
    private boolean     lastColumn          = false;    // current setting
    private boolean     lastRow             = false;    // current setting
    private boolean     parSpace            = false;    // current setting
    private boolean     priorEmptyPara      = false;    // current setting

    private Rectangle   aheadRect           = new Rectangle();
    private Rectangle   closeRect           = new Rectangle();
    private Rectangle   abackRect           = new Rectangle();

    private Color       pullColor           = Color.blue;       // current setting

    private Font        textFont            = null;     // current setting
    private FontMetrics textMetrics         = null;     // current setting
    private Color       textColor           = null;     // current setting
    private int         textHeight          = 0;        // current setting
    private int         textAlign           = LEFT;     // current setting
    private Rectangle   textRect            = null;     // current setting

    private int         colNum = 0, colWidth = 0, colSpacing = 0, numColumns = 0,
oneSetOff = 0;
    private int         nextWidth = 0, xb = 0, te = 0, xe = 0, ys = 0, yb = 0, ye = 0;

    public ezrpnlap(String readerContent, ez_text ez_Text, boolean readerFrame)
    {
    //  dbg("ezrpnlap.<init>");

        this.ez_Text = ez_Text;
        this.readerFrame = readerFrame;
        this.readerHeader = ez_Text.getTitle() != null;
        this.readerContent = readerContent;
        this.readerTextLength = readerContent.length();

        if (readerFrame)
        {
            this.setForeground(Color.black);
            this.setBackground(readerBackColor);
            stepFontSize(false);
        }
        else
```

```
        {
            this.setBackground(Color.white);    //  this is a hack
        }

        setAllFonts();
    }

    public synchronized void stepFontSize(boolean smaller)
    {
    //  dbg("ezrpnlap.stepFontSize(smaller = "+smaller+")");

        if (smaller)    readerSizeIndex--;
        else            readerSizeIndex++;

        readerHeadIndex = readerSizeIndex + 2;
        readerPullIndex = readerSizeIndex + 1;
        readerFootIndex = readerSizeIndex - 2;

        readerSizeIndex = Math.max(Math.min(readerSizeIndex,readerFontSize.length -
1),0);
        readerHeadIndex = Math.max(Math.min(readerHeadIndex,readerFontSize.length -
1),0);
        readerPullIndex = Math.max(Math.min(readerPullIndex,readerFontSize.length -
1),0);
        readerFootIndex = Math.max(Math.min(readerFootIndex,readerFontSize.length -
1),0);

        setAllFonts();
    }

    public synchronized boolean imageUpdate(Image img, int flags, int x, int y, int w,
int h)
    {
    //  dbg("ezrpnlap.imageUpdate(flags = "+flags+")");

        if (formattedText && (flags & ImageObserver.ALLBITS) != 0)  //  this image is
complete (at last)
        {
        //  resetReaderView();
            repaintReaderScreen();
            return false;
        }
        else
            return true;
    }

    public final synchronized void doViewerHome()
    {
    //  dbg("ezrpnlap.doViewerHome()");

        if (readerPageNum > 1)
        {
            readerPageNum = 1;
            readerStartIndex =
                ((Integer)readerPageStart.elementAt(
                    readerPageNum-1)).intValue();
        }
        repaintReaderScreen();
    }

    public final synchronized void doViewerLeft()
    {
    //  dbg("ezrpnlap.doViewerLeft()");

        if (readerPageNum > 1)
        {
            readerPageNum--;
            readerStartIndex =
                ((Integer)readerPageStart.elementAt(
                    readerPageNum-1)).intValue();
```

```
        }
        repaintReaderScreen();
    }

    public final synchronized void doViewerRight()
    {
    //  dbg("ezrpnlap.doViewerRight(): readerEndIndex = "+readerEndIndex);

        if (readerEndIndex < readerTextLength)
        {
            readerPageNum++;
            if (readerNumPages < readerPageNum)
            {
                readerStartIndex = readerEndIndex - 1;
                readerPageStart.addElement(
                    new Integer(readerStartIndex));
            }
            else    //  been there, done that
            {
                readerStartIndex =
                    ((Integer)readerPageStart.elementAt(
                        readerPageNum-1)).intValue();
            }
        }
        else if (readerNumPages == 0)
        {
            readerNumPages = readerPageNum;
        }
        repaintReaderScreen();
    }

    public final synchronized void doViewerSize(boolean smaller)
    {
    //  dbg("ezrpnlap.doViewerSize(boolean "+smaller+")");

        stepFontSize(smaller);
        setReaderRect();
        resetReaderView();
        repaintReaderScreen();
    }

    public synchronized void setAllFonts()
    {
    //  dbg("ezrpnlap.setAllFonts()");

        readerFooterFont = new Font("TimesRoman", Font.PLAIN,
readerFontSize[readerFootIndex]);
        readerScreenFont = new Font("TimesRoman", Font.PLAIN,
readerFontSize[readerSizeIndex]);
        readerPQuoteFont = new Font("TimesRoman", Font.PLAIN,
readerFontSize[readerPullIndex]);
        readerHeaderFont = new Font("TimesRoman", Font.PLAIN,
readerFontSize[readerHeadIndex]);
    }

    public synchronized void setReaderRect()
    {
    //  dbg("ezrpnlap.setReaderRect()");

        setReaderRect(this.bounds());
    }

    public synchronized void setReaderRect(Rectangle r)
    {
    //  dbg("ezrpnlap.setReaderRect("+r+")");

        readerRect.reshape
        (   1
        ,   1
        ,   r.width - 1
```

```
            ,   r.height - 1 - (readerFrame?readerFontSize[readerFootIndex]*3/2:0)
        );
    }

    public boolean mouseUp(Event evt, int x, int y)
    {
    //  dbg("ezrpnlap.mouseUp(Event "+evt+", int "+x+", int "+y+") ");

        if (!readerFrame) return false;
        else
        if (aheadRect.inside(x,y))
        {
            doViewerRight();
            return true;
        }
        else
        if (closeRect.inside(x,y))
        {
            ez_Text.disposeFrame();
            return true;
        }
        else
        if (abackRect.inside(x,y))
        {
            doViewerLeft();
            return true;
        }
        else return false;
    }

    public final Dimension minimumSize()
    {
    //  dbg("ezrpnlap.minimumSize()");

        return new Dimension(585, 400);
    }

    public final Dimension preferredSize()
    {
    //  dbg("ezrpnlap.preferredSize()");

        return minimumSize();
    }

    public synchronized void reshape(int x, int y, int width, int height)
    {
    //  dbg("ezrpnlap.reshape("+x+", "+y+", "+width+", "+height+")");

        lowestTextAreaBaseLine = 0;
        readerScreenDirty = true;
        resetReaderView();

    //  fillRect is performed here rather than in update(g)

        getGraphics().setColor(getBackground());
        getGraphics().fillRect(0, 0, width, height);
        getGraphics().setColor(getForeground());

        super.reshape(x, y, width, height);
    }

    public final synchronized void paint(Graphics g)
    {
    //  dbg("ezrpnlap.paint(g)");

        if (readerScreenDirty)
        {
            if (readerScreen != null)
                readerScreen.flush();
            readerScreen = this.createImage(this.size().width, this.size().height);
```

```
1
2               Graphics q = this.readerScreen.getGraphics();
3               paintReaderScreen(q);
4               q.dispose();
5               readerScreenDirty = false;
6           }
7
8           g.drawImage(readerScreen,0,0,this);
9       }
10
11      private final synchronized void paintReaderScreen(Graphics g)
12      {
13      //  dbg("ezrpnlap.paintReaderScreen(g)");
14
15          Rectangle r = bounds();
16
17          setReaderRect(r);                   //  set to full area (this.bounds())
18          g.setColor(getBackground());
19          g.fillRect(0, 0, r.width, r.height);
20
21          int numCols = calculateReaderColumns();
22          if (numCols != readerNumCols)
23          {
24              readerNumCols = numCols;
25              resetReaderView();
26          }
27
28          if (readerFrame)
29              readerLeading = 2;  //  make it more comfortable
30          else
31              readerLeading = 0;  //  make it more compact
32
33          if (readerHeader && !readerFrame)   //  draw header (article title)
34          {
35              lowestTextAreaBaseLine = 0;
36              g.setColor(Color.blue);
37              int titleEnd =  paintTextArea
38              (   g                   //  Graphics    g
39              ,   0                   //  int         beginIndex
40              ,   readerHeaderFont    //  Font        textFont
41              ,   ez_Text.getTitle()  //  String      textContent
42              ,   1                   //  int         numCols
43              ,   CENTER              //  int         alignment
44              );
45
46              lowestTextAreaBaseLine += readerHeaderFont.getSize()/2;
47              readerRect.reshape
48              (   1
49              ,   lowestTextAreaBaseLine
50              ,   r.width - 1
51              ,   r.height - lowestTextAreaBaseLine
52              );
53
54              g.setColor(getBackground());
55              g.fillRect(readerRect.x, readerRect.y, readerRect.width,
56  readerRect.height);
57          }
58
59          if (demo && readerFrame)
60          {
61              Font sFont = new Font
62              (   "Dialog"
63              ,   Font.BOLD
64              ,   2*readerFontSize[readerHeadIndex]
65              );
66              setFont
67              (   Color.white
68              ,   sFont
69              ,   g
70              );
```

```
1               String spam = "Demo";
2               int sWide = textMetrics.stringWidth(spam), i, j;
3               for
4               (   j = 2*readerPerimeter + textHeight/2
5               ;   j < readerRect.height
6               ;   j += 3*textHeight/2
7               )
8                   for
9                   (   i = 4*readerPerimeter + textHeight/2 - j
10                  ;   i < readerRect.width
11                  ;   i += 3*sWide/2
12                  )
13                      g.drawString    //  draw the spam
14                      (   spam
15                      ,   i + readerRect.x
16                      ,   j + readerRect.y
17                      );
18          }
19
20          g.setColor(getForeground());
21
22          readerEndIndex = paintTextArea(g, readerStartIndex);
23
24          if (readerNumPages == 0)
25          {
26              if (readerEndIndex == readerTextLength)
27                  readerEstPages = readerNumPages = readerPageNum;
28              else    //  estimate number of pages
29              {
30                  readerEstPages =
31  Math.round(0.5f+(float)readerPageNum*(float)readerTextLength/(float)readerEndIndex);
32              }
33          }
34          else
35              readerEstPages = readerNumPages;
36
37          if (readerFrame)    //  draw footer (page information)
38          {
39              StringBuffer pI = new StringBuffer();
40              pI.append("Screen ").append(String.valueOf(readerPageNum)).append(" of ");
41              if (readerNumPages == 0) pI.append("~");
42              pI.append(String.valueOf(readerEstPages));
43              String pageInfo = pI.toString();
44
45              int infoWidth = getFontMetrics(readerFooterFont).stringWidth(pageInfo);
46
47              g.setFont(readerFooterFont);
48              g.setColor(getForeground());
49              int xLoc = (r.width - infoWidth)/2;
50              int yLoc =  r.height - readerFontSize[readerFootIndex]/4 - 1;
51              g.drawString(pageInfo, xLoc, yLoc);
52          }
53      }
54
55      private final synchronized int paintTextArea
56      (   Graphics    g
57      ,   int         beginIndex
58      )                                                       //  returns string
59  index of first non-drawn character
60      {
61      //  dbg("ezrpnlap.paintTextArea(g, "+beginIndex+"): "+readerNumCols+" columns");
62
63          return paintTextArea
64          (   g
65          ,   beginIndex
66          ,   readerScreenFont
67          ,   this.readerContent
68          ,   readerNumCols
69          ,   readerAlignment
70          );
```

```
    )

    private final synchronized int paintTextArea
    (   Graphics    g
    ,   int         beginIndex
    ,   Font        theFont
    ,   String      textContent
    ,   int         numCols
    ,   int         alignment
    )                                                       //  returns string
index of first non-drawn character
    {
    //  dbg("ezrpnlap.paintTextArea()");                    //
    paintTextArea(\n(g\n, "+beginIndex+"\n "+theFont+"\n, textContent\n "+numCols+"\n,
"+alignment+"\n)

        int endText     = textContent.length();
        int endIndex    = beginIndex;
        boolean go = endIndex != endText;
        numColumns = numCols;

        textAlign = alignment;
        textColor = g.getColor();

        this.setFont(textColor,theFont,g);

        StringTokenizer paragraphs = new StringTokenizer        //  break up the text
into normal paragraphs
            (textContent.substring(beginIndex),"<",true);       //  final true returns
token on next call

        colNum = 0;
        int colMargin = 4*textMetrics.stringWidth(BL);          //  four spaces
between cols
        colWidth = (readerRect.width - 2*readerPerimeter - (numColumns-
1)*colMargin)/numColumns;
        colSpacing = colWidth + colMargin;

        String nextWord = MT;
        StringBuffer thisLine = new StringBuffer(70);

        xb = readerRect.x + readerPerimeter;                    //  local left edge of
the column area
        te = xb;                                                //  end position of
this line's words
        xe = xb + colWidth;                                     //  local right edge
of the column area

        nextWidth = 0;

        ys = readerRect.y + (textHeight + readerLeading)*3/4;   //  local starting
baseline of first line
        yb = ys;
        ye = readerRect.y + readerRect.height - textHeight/4;   //  local maximum
baseline of last line
        lowestTextAreaBaseLine  = yb;                           //  lowest baseline in
readerRect
        priorEmptyPara = false;                                 //  permits paragraph
spacers
        lastColumn = colNum == numColumns - 1;
        lastRow = false;
        controlSize = textHeight*2/3;
        oneSetOff = (textHeight + readerLeading)/2;             //  use stnd font size
for half-space set-offs
    //  dbg("... oneSetOff = "+oneSetOff);

        abackRect.reshape
        (   readerRect.x + readerRect.width - 2 - (controlSize+4)*3
        ,   readerRect.y + readerRect.height - 4 - controlSize
        ,   controlSize + 4
```

```
1           ,   controlSize + 4
2           );
3           if (debug)
4   g.drawRect(abackRect.x,abackRect.y,abackRect.width,abackRect.height);
5
6           closeRect.reshape
7           (   readerRect.x + readerRect.width - 2 - (controlSize+4)*2
8           ,   readerRect.y + readerRect.height - 4 - controlSize
9           ,   controlSize + 4
10          ,   controlSize + 4
11          );
12          if (debug)
13  g.drawRect(closeRect.x,closeRect.y,closeRect.width,closeRect.height);
14
15          aheadRect.reshape
16          (   readerRect.x + readerRect.width - 2 - (controlSize+4)*1
17          ,   readerRect.y + readerRect.height - 4 - controlSize
18          ,   controlSize + 4
19          ,   controlSize + 4
20          );
21          if (debug)
22  g.drawRect(aheadRect.x,aheadRect.y,aheadRect.width,aheadRect.height);
23
24          draw: while (paragraphs.hasMoreTokens() && go)          //  paragraph to
25  follow
26          {
27              String para = paragraphs.nextToken("<");
28
29          //  dbg("... new paragraph token = ["+para+"]");
30
31              if (formattedText
32                  && para.startsWith("`"))                        //  test for in-line
33  title
34              {
35              //  dbg("... pull quote: yb = "+yb);
36
37                  rollBack();
38
39                  int pullQuoteHeight = 2*pullQuoteLineWidth      //  reserve pull-quote
40  bars
41                      + oneSetOff*((yb <= ys)?1:2);               //  reserve 1 or 2
42  setOffs outside bars
43
44                  textAlign = CENTER;
45                  this.setFont(Color.blue,readerPQuoteFont,g);
46
47                  pullQuoteHeight += textHeight + readerLeading;  //  reserve one
48  pullquote line of text
49              //  dbg("... pullQuoteHeight = "+pullQuoteHeight);
50
51                  if (yb + pullQuoteHeight > readerRect.y + readerRect.height)
52                  {                                               //  not enough room in
53  this (partial?) column
54                      if (lastColumn) break draw;                 //  no more columns
55  left to fill
56                      nextColumn();
57                      yb = readerRect.y;                          //  start at the top
58                  }
59                  else if (yb > ys)                               //  not at the top of
60  the column?
61                  {
62                      yb += oneSetOff;                            //  upper set-off
63                  }
64
65                  for (int line = 0; line < pullQuoteLineWidth; line++)
66                  {
67                      g.drawLine(xb, yb, xe, yb++);
68                  }
69                  yb += textHeight + readerLeading;               //  drop for next line
70                  drawPullUnderliner = finishPullQuoteText = true;
```

```
1
2                   para = para.substring(1);                       //  drop control
3   character
4                   endIndex++;                                     //  update location
5               }
6               else if (formattedText
7                    && drawPullUnderliner && !finishPullQuoteText) //  restore stnd
8   values
9               {
10                  rollBack();
11
12                  for (int line = 0; line < pullQuoteLineWidth; line++)
13                  {
14                      g.drawLine(xb, yb, xe, yb++);
15                  }
16                  drawPullUnderliner = false;
17
18                  textAlign = alignment;
19                  this.setFont(textColor,theFont,g);
20
21                  if (yb > ys)
22                      yb += oneSetOff
23                            + textHeight + readerLeading;         //  drop for next line
24                  if (yb > ye)
25                  {                                               //  not enough room in
26  this (partial?) column
27                      if (lastColumn) break draw;                 //  no more columns
28  left to fill
29                      nextColumn();
30                  }
31              }
32
33              if (para.equals("<"))                               //  an html new
34  paragraph?
35              {
36                  String tagText = paragraphs.nextToken(">").toLowerCase();
37                  int tagLength = tagText.length();
38
39                  para = paragraphs.nextToken();
40                  if (para.equals(">"))
41                  {
42                      if (tagText.startsWith("p"))
43                      {
44                          if (yb > ys && priorEmptyPara)          //  unless we're at
45  the top
46                          {
47                              yb += textHeight + readerLeading;   //  drop down to
48  another line
49                              if (yb > ye)
50                              {                                   //  not enough room in
51  this (partial?) column
52                                  if (lastColumn) break draw;     //  no more columns
53  left to fill
54                                  nextColumn();
55                              }
56                          }
57                      }
58                      lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);
59                      priorEmptyPara = true;
60                  }
61                  endIndex += tagLength+2;
62              }
63              else if (formattedText
64                   && para.startsWith("@")
65                   && (   para.toLowerCase().endsWith(".jpeg")
66                       ||  para.toLowerCase().endsWith(".gif")))  //  test for in-line
67  graphic
68              {
69                  Image theImage = ez_Text.fetchImage(para.substring(1).trim());
70
```

```
                if (theImage != null)
                {
                    rollBack();

                    priorEmptyPara = false;

                    int wIm = theImage.getWidth(ez_Text);
                    int hIm = theImage.getHeight(ez_Text);
                    if (wIm > colWidth)                         //  scale graphic down
to column width
                    {
                        hIm = Math.round( (float)colWidth * (float)hIm / (float)wIm );
                        wIm = colWidth;
                    }

                    if (hIm > readerRect.height)                //  scale graphic down
to column height
                    {
                        wIm = Math.round( (float)readerRect.height * (float)wIm /
(float)hIm );
                        hIm = readerRect.height;
                    }

                    int fullImageHeight = hIm + ((yb > ys)?oneSetOff:0);
                    if (yb + fullImageHeight > readerRect.y + readerRect.height)
                    {                                           //  not enough room in
this (partial?) column
                        if (lastColumn) break draw;             //  no more columns
left to fill
                        nextColumn();
                        rollBack();
                    }
                    else if (yb > ys)
                    {
                        yb += oneSetOff;                        //  upper spacing
                    }

                    int xIm = xb + (colWidth - wIm)/2;          //  center it

                    g.fillRect(xIm, yb, wIm, hIm);
                    g.drawImage(theImage, xIm, yb, wIm, hIm, this);
                    endIndex += para.length();                  //  update location
                    yb += hIm;                                  //  image sizing
                    lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);

                    yb += oneSetOff
                            + textHeight + readerLeading;       //  image spacing
                    if (yb  > ye)                               //  no room for text
under image margin
                    {
                        if (lastColumn) break draw;             //  no more columns
left to fill
                        nextColumn();
                    }
                }
            }
            else                                                //  none of the
above...
            {
            //  dbg("... new paragraph text:
"+para.substring(0,Math.min(20,para.length())));

                priorEmptyPara = false;

                if (formattedText && finishPullQuoteText)
                {
                    textAlign = CENTER;
                    this.setFont(Color.blue, readerPQuoteFont, g);
                }

```

```
                te = xb;                                        //  start a new line
                StringTokenizer words =
                    new StringTokenizer(para,BL,true);          //  "true" returns
white spaces

                while (words.hasMoreTokens())
                {
                    nextWord = words.nextToken();
                    nextWidth = textMetrics.stringWidth(nextWord);

                    if (te + nextWidth < xe)                    //  it will fit on
this line, so add it on
                    {
                        thisLine.append(nextWord);
                        te += nextWidth;
                        nextWord = MT;
                        nextWidth = 0;
                    }
                    else                                        //  it won't fit if
added, so draw the text
                    {
                        if (thisLine.length() > 0)              //  not if a monster
is first in the article
                        {
                        //  dbg("new line = ["+thisLine+"], nextWord =
["+nextWord+"]");
                            endIndex += this.drawText(thisLine, g);
                            thisLine.setLength(0);              //  start the new line
                        }

                        if (yb > ye)                            //  start a new
column?
                        {
                            if (lastColumn) break draw;         //  no more columns
left to fill
                            nextColumn();

                            while (thisLine.toString().startsWith(BL))
                            {                                   //  ignore preceeding
blanks on new line
                                String noBlank = thisLine.toString().substring(1);
                                thisLine.setLength(0);
                                thisLine.append(noBlank);
                                endIndex++;
                            }
                        }

                        if (thisLine.length() == 0)
                        while (nextWord.startsWith(BL))         //  ignore preceeding
blanks on new line
                        {
                            nextWord = nextWord.substring(1);   //  look again
                            nextWidth = textMetrics.stringWidth(nextWord);
                            endIndex++;
                        }
                        thisLine.append(nextWord);
                        te = xb + nextWidth;                    //  possibly a
monster!
                        nextWord = MT;
                        nextWidth = 0;

                        while (te > xe)                         //  shorten the line
by breaking the word
                        {
                        //  dbg("it's big: thisLine = ["+thisLine+"]");
                            while (te > xe)                     //  transfer letters
back until fit (one line)
                            {
                                int e = thisLine.length();
```

```
                                nextWord = thisLine.toString().substring(e-1,e) +
nextWord;
                                String truncated = thisLine.toString().substring(0,e-
1);
                                thisLine.setLength(0);
                                thisLine.append(truncated);
                            //  dbg(" - try thisLine = ["+thisLine+"]\n...and nextWord
= ["+nextWord+"]");
                                te = xb +
textMetrics.stringWidth(thisLine.toString()+"-");
                            }
                            thisLine.append("-");
                            endIndex += this.drawText(thisLine, g) - 1;
                            thisLine.setLength(0);              //  start the new line

                            if (yb > ye)                        //  start a new
column?
                            {
                                if (lastColumn) break draw;     //  no more columns
left to fill
                                nextColumn();
                            }
                            thisLine.append(nextWord);
                            te = xb + textMetrics.stringWidth(thisLine.toString());
                            nextWord = MT;
                            nextWidth = 0;
                        }
                    }
                }                                               //  no more words in
this paragraph
                if (thisLine.length() > 0)
                    endIndex += this.drawText(thisLine, g);
                thisLine.setLength(0);                          //  start the new line

                if (yb > ye)                                    //  start a new
column?
                {
                    if (lastColumn) break draw;                 //  no more columns
left to fill
                    nextColumn();
                }
                nextWord = MT;                                  //  starting fresh
                nextWidth = 0;
                te = xb;                                        //  pixel width of
this line's words
            }
            finishPullQuoteText = false;
        }   //  end of draw label block                         //  no more paragraphs

        while (endIndex < endText
        && textContent.substring(endIndex++).startsWith(BL));   //  skip blanks

        g.setColor(getForeground());
    //  left arrow
        if (readerPageNum > 1)
        {
            int x = abackRect.x + 2
            ,   y = abackRect.y + abackRect.height/2;
            for (int arrow = 0; arrow < controlSize; arrow++)
            {
                g.drawLine
                (   x + arrow
                ,   y - arrow/2
                ,   x + arrow
                ,   y + arrow/2
                );
            }
        }
    //  close box
        if (readerFrame)
```

```
        {
            int x = closeRect.x + 2
            ,   y = closeRect.y + 2
            ,   w = closeRect.width  - 5
            ,   h = closeRect.height - 5;

            g.drawRect(x, y, w, h);
            g.drawLine(x, y, x+w, y+h);
            g.drawLine(x+w, y, x, y+h);
        }
    //  right arrow
        if (endIndex < endText)
        {
            int x = aheadRect.x + aheadRect.width - 2
            ,   y = abackRect.y + abackRect.height/2;
            for (int arrow = 0; arrow < controlSize; arrow++)
            {
                g.drawLine
                (   x - arrow
                ,   y + arrow/2
                ,   x - arrow
                ,   y - arrow/2
                );
            }
        }
        return endIndex;
    }

    private final synchronized void resetReaderView()
    {
    //  dbg("ezrpnlap.resetReaderView()");

        readerStartIndex    = 0;
        readerEndIndex      = 0;
        readerNumPages      = 0;
        readerEstPages      = 1;
        readerPageNum       = 1;
        readerPageStart.removeAllElements();
        readerPageStart.addElement(new Integer(readerStartIndex));
    }

    public final void dispose()
    {
    //  dbg("ezrpnlap.dispose()");

        if (readerScreen != null)
            readerScreen.flush();
    }

    public synchronized void update(Graphics g)
    {
    //  dbg("ezrpnlap.update()");

    //  fillRect is NOT performed here to eliminate blank screen boredom during
offscreen drawing

    //  g.setColor(getBackground());
    //  g.fillRect(0, 0, width, height);
    //  g.setColor(getForeground());

        paint(g);
    }

    private final synchronized void dbg(String  s)
    {
        if (debug) System.out.println(s);
    }
    private boolean debug = false;      //  Print debugging info?

    private final synchronized void repaintReaderScreen()
```

```
    {
    //  dbg("ezrpnlap.repaintReaderScreen()");

        readerScreenDirty = true;
        repaint();
    }

    private final synchronized int calculateReaderColumns()
    {
    //  dbg("ezrpnlap.calculateReaderColumns()");

        int numCols = 1;

        Rectangle r = bounds(); //  rectangle of ezrpnlap
        int numChar = 30;       //  the control parameter: min # of n's per line of
text
        int targetWidth = numChar * getFontMetrics(readerScreenFont).stringWidth("n");

        setReaderRect(r);
        numCols = (r.width)/targetWidth;

        if (numCols < 1)
            numCols = 1;

        return numCols;
    }

    private final synchronized int drawText(StringBuffer thisLine, Graphics g)
    {
    //  dbg("ezrpnlap.drawText(\""+thisLine.toString()+"\")");

        g.drawString
        (   thisLine.toString()
        ,   xb + textAlign*(colWidth + xb - te)/2
        ,   yb
        );                                              //  draw the text
        lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);
        yb += textHeight + readerLeading;               //  drop down to another line
        checkShortLine();

        return thisLine.length();
    }

    private final synchronized void checkShortLine()
    {
    //  dbg("ezrpnlap.checkShortLine()");

        if (lastColumn
        && (lastRow = (ye - yb) <= (textHeight + readerLeading)*2))
        {
            if (readerPageNum > 1)                      //  left arrow
                xe = abackRect.x;
            else if (readerFrame)                       //  close box
                xe = closeRect.x;
            else                                        //  right arrow
                xe = aheadRect.x;
        }

    //  dbg("ezrpnlap.checkShortLine(): new xe = "+xe);
    }

    private final synchronized void nextColumn()
    {
    //  dbg("ezrpnlap.nextColumn()");

        colNum++;                                       //  finished another column
        lastColumn = colNum == numColumns - 1;
    //  clearInitialSpaces = true;

```

```
1           yb = ys;                                        //  local baseline of first
2   line
3           xb += colSpacing;                               //  local left edge of the
4   column area
5           te += colSpacing;                               //  local end of text
6           xe += colSpacing;                               //  local right edge of the
7   column area
8
9           checkShortLine();
10      }
11
12      private final synchronized void setFont(Font f, Graphics g)
13      {
14      //  dbg("ezrpnlap.setFont("+f+")");
15
16          textMetrics = getFontMetrics(f);
17          textHeight = f.getSize()*5/4;
18          g.setFont(f);
19
20      //  dbg("ezrpnlap.setFont(setFont): textHeight = "+textHeight);
21      }
22
23      private final synchronized void setFont(Color c, Font f, Graphics g)
24      {
25      //  dbg("ezrpnlap.("+c+", "+f+", g)");
26
27          g.setColor(c);
28          setFont(f, g);
29      }
30
31      private final synchronized void rollBack()
32      {
33      //  dbg("ezrpnlap.rollBack()");
34
35          if (yb == ys && drawPullUnderliner && !finishPullQuoteText)
36          {
37              yb = readerRect.y
38                      + readerRect.height
39                      - pullQuoteLineWidth;               //  roll back to base of
40  previous column
41              colNum--;
42              lastColumn = false;
43
44              xb -= colSpacing;                           //  local left edge of the
45  column area
46              te -= colSpacing;                           //  local end of text
47              xe -= colSpacing;                           //  local right edge of the
48  column area
49          }
50          else if (yb > ys)
51          {
52              yb -= (textHeight + readerLeading)/2;       //  roll back half of last CR
53          }
54          else
55          {
56              yb = readerRect.y;                          //  start at the top
57          }
58      }
59  }
60
61  /**
62   *  sample web page with applet tag
63   */
64  <HTML>
65  <HEAD>
66    <META NAME="GENERATOR" CONTENT="Adobe PageMill 2.0 Mac">
67    <TITLE></TITLE>
68  </HEAD>
69  <BODY>
70
```

```
<applet codebase="ez_class" code="ez_text.class" width=300 height=200>
<param name="fileName" value="ezsample.txt">
</applet>

</BODY>
</HTML>

/**
 *  ezsample.txt web page for applet
 */
`EZ Text Sample Article
`"Click on the applet"
The reader screen presentation of content is interactive and optimized for
readability. Font size can be changed by pressing up or down arrow keys, or by
clicking the Font+ or Font- buttons. Pages can be turned by pressing home, left, or
right keys, or by clicking the Home, Previous, or Next buttons.
`"optimized for readability"
One of the key advantages of this reader screen presentation is its careful avoidance
of a particular form of eye-strain known as optokinetic nystagmus. Nystagmus, a
reflexive jump in the muscles directing the eye, is triggered by, for example,
vertical scrolling of text on a computer screen.
@ez_eye.gif
As nystagmus strain increases, muscle jumps can become exaggerated and even erratic,
finally resulting in a perception of visual weariness. Nystagmus is well-known as a
side-effect of excessive alcohol: a simple road-side test conducted by law officers
can aid in determination of a possible DWI offense. Other possible causes of nystagmus
include general physical exhaustion and illness.

While the term "nystagmus" may have been unfamiliar, the condition is encountered in
many circumstances, none of which is very pleasant.
`"optokinetic nystagmus"
The main side effect of optokinetic nystagmus induced by scrolling text is a tendency
to print the document, so that it can be read later with greater comfort. This
interrupts delivery of content and consumes resources. It is very possible that the
demise of the paperless office dreams of the '80's was caused in part by nystagmus.
Another major cause of its demise was the difference in formatting between standard
paper, height being typically 30% greater than width, and standard computer monitors,
width being typically 33% greater than height. Word processors targeted paper
formatting, so that documents delivered from computer to computer were not easily
adapted to screen formats. This pattern also resulted in a tendency to print such
documents rather than struggle to consume their contents on screen.

EZ Text solves these twin problems nicely. First, there is no scrolling text,
resulting in reduced eye-strain. Font sizing can be adjusted by the reader for his
particular situation of lighting, monitor quality and size, and his personal visual
comfort. Second, The content is automatically laid out in columns of text, with
neither too few words in a line, about 30 characters minimum, nor too many, about 60
characters maximum. The upper limit yields a line length which, as the human eye
executes its rapid raster motion to the beginning of the next line, is not too long to
accurately and easily find the correct next new line. If lines are much shorter, below
the minimum, lines of text contain too little content, and the frequent raster motions
can induce eye-strain.
`"Web pages ... become a rewarding experience with EZ Text."
This sample document contains about 3300 characters in about 640 words. At normal
reading speeds, it should take about 5 minutes to absorb its main content. The effort
required to read this article on the screen, using EZ Text, should be noticeably less
than required by other channels. For instance, using a word processor, two approaches
are common. The first is to print it out, collect the page(s), and finally read it.
Alternatively, one could open the document in a word processor, adjust window widths
and zoom factors, and finally read it, occasionally scrolling to bring unseen parts
into view. With EZ Text, just sit back, tap the up arrow to adjust the font, and tap
the right arrow to turn the page as needed. Web pages, known widely as challenging
when it comes to delivering on-screen content, can become a rewarding experience with
EZ Text.

```

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☑ BLURED OR ILLEGIBLE TEXT OR DRAWING
- ☑ SKEWED/SLANTED IMAGES
- ☐ COLORED OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☑ REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning documents *will not* correct images problems checked, please do not report the problems to the IFW Image Problem Mailbox